Spring Boot + Redis

Redis

Redis 采用内存（in-memory）数据集（Dataset），根据使用场景，可以通过每隔一段时间转储数据集到磁盘，或者追加每条命令到日志来持久化。持久化也可以被禁用，如果你只是需要一个功能丰富、网络化的内存缓存。

数据模型

- String（字符串）
- Hash（哈希）
- List（列表）
- Set（集合）
- Zset（Sorted Set：有序集合）

关键优势

- Redis 的优势包括它的速度、对富数据类型的支持、操作的原子性，以及通用性：
- 性能极高，它每秒可执行约 100,000 个 Set 以及约 100,000 个 Get 操作；
- 丰富的数据类型，Redis 对大多数开发人员已知的大多数数据类型提供了原生支持，这使得各种问题得以轻松解决；
- 原子性，因为所有 Redis 操作都是原子性的，所以多个客户端会并发地访问一个 Redis 服务器，获取相同的更新值；

spring-boot-starter-data-redis

spring-boot-starter-data-redis Spring Boot 1.0 默认使用的是 Jedis 客户端，2.0 替换成了 Lettuce

- Lettuce：是一个可伸缩线程安全的 Redis 客户端，多个线程可以共享同一个 RedisConnection，它利用优秀 Netty NIO 框架来高效地管理多个连接。
- Spring Data：是 Spring 框架中的一个主要项目，目的是为了简化构建基于 Spring 框架应用的数据访问，包括非关系数据库、Map-Reduce 框架、云数据服务等，另外也包含对关系数

据库的访问支持。
- Spring Data Redis：是 Spring Data 项目中的一个主要模块，实现了对 Redis 客户端 API 的高度封装，使对 Redis 的操作更加便捷。

快速上手

- 依赖

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-pool2</artifactId>
</dependency>
```

引入 commons-pool 2 是因为 Lettuce 需要使用 commons-pool 2 创建 Redis 连接池。

- application 配置

```
# Redis 数据库索引（默认为 0）
spring.redis.database=0
# Redis 服务器地址
spring.redis.host=localhost
# Redis 服务器连接端口
spring.redis.port=6379
# Redis 服务器连接密码（默认为空）
spring.redis.password=
# 连接池最大连接数（使用负值表示没有限制） 默认 8
spring.redis.lettuce.pool.max-active=8
# 连接池最大阻塞等待时间（使用负值表示没有限制） 默认 -1
spring.redis.lettuce.pool.max-wait=-1
# 连接池中的最大空闲连接 默认 8
spring.redis.lettuce.pool.max-idle=8
# 连接池中的最小空闲连接 默认 0
spring.redis.lettuce.pool.min-idle=0
```

基础操作

- 存取实体

```
@Test
public void testObj(){
    User user=new User("ityouknow@126.com", "smile", "youknow", "know","2020");
    ValueOperations<String, User> operations=redisTemplate.opsForValue();
    operations.set("com.neo", user);
    User u=operations.get("com.neo");
    System.out.println("user: "+u.toString());
}
```

- 超时失效

```
@Test
public void testExpire() throws InterruptedException {
    User user=new User("ityouknow@126.com", "expire", "youknow", "expire","2020");
    ValueOperations<String, User> operations=redisTemplate.opsForValue();
    operations.set("expire", user,100,TimeUnit.MILLISECONDS);
    Thread.sleep(1000);
    boolean exists=redisTemplate.hasKey("expire");
    if(exists){
        System.out.println("exists is true");
    }else{
        System.out.println("exists is false");
    }
}
```

- 删除数据

```
@Test
public void testDelete() {
    ValueOperations<String, User> operations=redisTemplate.opsForValue();
    redisTemplate.opsForValue().set("deletekey", "ityouknow");
    redisTemplate.delete("deletekey");
    boolean exists=redisTemplate.hasKey("deletekey");
    if(exists){
        System.out.println("exists is true");
    }else{
        System.out.println("exists is false");
    }
}
```

- Hash

```
@Test
```

```
public void testHash() {
    HashOperations<String, Object, Object> hash = redisTemplate.opsForHash();
    hash.put("hash","you","you");
    String value=(String) hash.get("hash","you");
    System.out.println("hash value :"+value);
}
```

- List

```
@Test
public void testList() {
    ListOperations<String, String> list = redisTemplate.opsForList();
    list.leftPush("list","it");
    list.leftPush("list","you");
    list.leftPush("list","know");
    String value=(String)list.leftPop("list");
    System.out.println("list value :"+value.toString());
}

List<String> values=list.range("list",0,2);
   for (String v:values){
       System.out.println("list range :"+v);
   }
```

> Redis List 的实现为一个双向链表，即可以支持反向查找和遍历，更方便操作，不过带来了部分额外的内存开销，Redis 内部的很多实现，包括发送缓冲队列等也都是用的这个数据结构。

- Set

Redis Set 对外提供的功能与 List 类似是一个列表的功能，特殊之处在于 Set 是可以自动排重的，当你需要存储一个列表数据，又不希望出现重复数据时，Set 是一个很好的选择，并且 Set 提供了判断某个成员是否在一个 Set 集合内的重要接口，这个也是 List 所不能提供的。

```
@Test
public void testSet() {
    String key="set";
    SetOperations<String, String> set = redisTemplate.opsForSet();
    set.add(key,"it");
    set.add(key,"you");
    set.add(key,"you");
    set.add(key,"know");
    Set<String> values=set.members(key);
    for (String v:values){
```

```
        System.out.println("set value :"+v);
    }
}
```

实用方法：difference，unions

- ZSet

Redis Sorted Set 的使用场景与 Set 类似，区别是 Set 不是自动有序的，而 Sorted Set 可以通过用户额外提供一个优先级（Score）的参数来为成员排序，并且是插入有序，即自动排序。

```
@Test
public void testZset(){
    String key="zset";
    redisTemplate.delete(key);
    ZSetOperations<String, String> zset = redisTemplate.opsForZSet();
    zset.add(key,"it",1);
    zset.add(key,"you",6);
    zset.add(key,"know",4);
    zset.add(key,"neo",3);

    Set<String> zsets=zset.range(key,0,3);
    for (String v:zsets){
        System.out.println("zset value :"+v);
    }

    Set<String> zsetB=zset.rangeByScore(key,0,3);
    for (String v:zsetB){
        System.out.println("zsetB value :"+v);
    }
}
```

实用Redis 实现 Session 同步

目前主流的分布式 Session 管理有两种方案。

- Session 复制

在微服务架构中，往往需要 N 个服务端来共同支持服务，不建议采用这种方案。

- Session 集中存储

在单独的服务器或服务器集群上使用缓存技术，如 Redis 存储 Session 数据，集中管理所有的 Session，所有的 Web 服务器都从这个存储介质中存取对应的 Session，实现 Session 共享。将

Session 信息从应用中剥离出来后，其实就达到了服务的无状态化，这样就方便在业务极速发展时水平扩充。

> Session 与 Cookie
> - Session 和 Cookie 都是记录客户状态的机制
> - Cookie 保存在客户端浏览器中
> - Session 保存在服务器上

Spring Session

Spring Session 提供了一套创建和管理 Servlet HttpSession 的方案。Spring Session 提供了集群 Session（Clustered Sessions）功能，默认采用外置的 Redis 来存储 Session 数据，以此来解决 Session 共享的问题。

- API 和用于管理用户会话的实现；
- HttpSession，允许以应用程序容器（即 Tomcat）中性的方式替换 HttpSession；
- 将 Session 所保存的状态卸载到特定的外部 Session 存储中，如 Redis 或 Apache Geode 中，它们能够以独立于应用服务器的方式提供高质量的集群；
- 支持每个浏览器上使用多个 Session，从而能够很容易地构建更加丰富的终端用户体验；
- 控制 Session ID 如何在客户端和服务器之间进行交换，这样的话就能很容易地编写 Restful API，因为它可以从 HTTP 头信息中获取 Session ID，而不必再依赖于 cookie；
- 当用户使用 WebSocket 发送请求的时候，能够保持 HttpSession 处于活跃状态。

需要说明的很重要的一点就是，Spring Session 的核心项目并不依赖于 Spring 框架，因此，我们甚至能够将其应用于不使用 Spring 框架的项目中。

spring-session-data-redis

- 引入依赖包

```
<dependency>
    <groupId>org.springframework.session</groupId>
    <artifactId>spring-session-data-redis</artifactId>
</dependency>
```

- 配置文件

```
# 数据库配置
spring.datasource.url=jdbc:mysql://localhost:3306/test?serverTimezone=UTC&useUnicode=true&characterEncoding=utf-8&useSSL=true
```

```
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
# JPA 配置
spring.jpa.properties.hibernate.hbm2ddl.auto=create
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5InnoDBDialect
spring.jpa.show-sql= true
# Redis 配置
# Redis 数据库索引（默认为0）
spring.redis.database=0
# Redis 服务器地址
spring.redis.host=localhost
# Redis 服务器连接端口
spring.redis.port=6379
# Redis 服务器连接密码（默认为空）
spring.redis.password=
# 连接池最大连接数（使用负值表示没有限制）
spring.redis.lettuce.pool.max-active=8
spring.redis.lettuce.pool.max-wait=-1
spring.redis.lettuce.shutdown-timeout=100
spring.redis.lettuce.pool.max-idle=8
spring.redis.lettuce.pool.min-idle=0
```

在项目中创建 SessionConfig 类，使用注解配置其过期时间。

- Session 配置：

```
@Configuration
@EnableRedisHttpSession(maxInactiveIntervalInSeconds = 86400*30)
public class SessionConfig {
}
```

- maxInactiveIntervalInSeconds: 设置 Session 失效时间，使用 Redis Session 之后，原 Spring Boot 中的 server.session.timeout 属性不再生效。

Spring Boot 中使用 Cache 缓存的使用

Spring Cache 利用了 Spring AOP 的动态代理技术，在项目启动的时候动态生成它的代理类，在代理类中实现了对应的逻辑。

Spring 的缓存技术还具备相当的灵活性，不仅能够使用 SpEL（Spring Expression Language）来定义缓存的 key 和各种 condition，还提供了开箱即用的缓存临时存储方案，也支持和主流的专业缓存如 EHCache 集成。

> SpEL（Spring Expression Language）是一个支持运行时查询和操作对象图的强大的表达式语言，其语法类似于统一 EL，但提供了额外特性，显式方法调用和基本字符串模板函数。

特点

- 通过少量的配置 Annotation 注释即可使得既有代码支持缓存；
- 支持开箱即用 Out-Of-The-Box，即不用安装和部署额外第三方组件即可使用缓存；
- 支持 Spring Express Language，能使用对象的任何属性或者方法来定义缓存的 key 和 condition；
- 支持 AspectJ，并通过其实现任何方法的缓存支持；
- 支持自定义 key 和自定义缓存管理者，具有相当的灵活性和扩展性。

spring-boot-starter-cache

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-cache</artifactId>
</dependency>
```

- @Cacheable

@Cacheable 用来声明方法是可缓存的，将结果存储到缓存中以便后续使用相同参数调用时不需执行实际的方法，直接从缓存中取值。@Cacheable 可以标记在一个方法上，也可以标记在一个类上。当标记在一个方法上时表示该方法是支持缓存的，当标记在一个类上时则表示该类所有的方法都是支持缓存的。

```
- value：缓存的名称。
- key：缓存的 key，可以为空，如果指定要按照 SpEL 表达式编写；如果不指定，则缺省按照方法的所有参数进行组合。
- condition：触发条件，只有满足条件的情况才会加入缓存，默认为空，既表示全部都加入缓存，支持 SpEL。
```

- @CachePut

与 @Cacheable 不同的是使用 @CachePut 标注的方法在执行前，不会去检查缓存中是否存在之前执行过的结果，而是每次都会执行该方法，并将执行结果以键值对的形式存入指定的缓存中。

```
- value 缓存的名称。
- key 缓存的 key，可以为空，如果指定要按照 SpEL 表达式编写，如果不指定，则缺省按照方法的所
```

有参数进行组合。
- condition 缓存的条件，可以为空，使用 SpEL 编写，返回 true 或者 false，只有为 true 才进行缓存。

- @CacheEvict

@CacheEvict 是用来标注在需要清除缓存元素的方法或类上的，当标记在一个类上时表示其中所有的方法的执行都会触发缓存的清除操作。@CacheEvict 可以指定的属性有 value、key、condition、allEntries 和 beforeInvocation，其中 value、key 和 condition 的语义与 @Cacheable 对应的属性类似。

即 value 表示清除操作是发生在哪些 Cache 上的（对应 Cache 的名称）；key 表示需要清除的是哪个 key，如未指定则会使用默认策略生成的 key；condition 表示清除操作发生的条件。下面来介绍一下新出现的两个属性 allEntries 和 beforeInvocation。

allEntries 属性
allEntries 是 boolean 类型，表示是否需要清除缓存中的所有元素，默认为 false，表示不需要。当指定了 allEntries 为 true 时，Spring Cache 将忽略指定的 key，有的时候我们需要 Cache 一下清除所有的元素，这比一个一个清除元素更有效率。

beforeInvocation 属性
清除操作默认是在对应方法成功执行之后触发的，即方法如果因为抛出异常而未能成功返回时也不会触发清除操作。使用 beforeInvocation 可以改变触发清除操作的时间，当我们指定该属性值为 true 时，Spring 会在调用该方法之前清除缓存中的指定元素。